

Advanced modelling of qualitative networks with BMA

<http://biomodelanalyzer.org/>

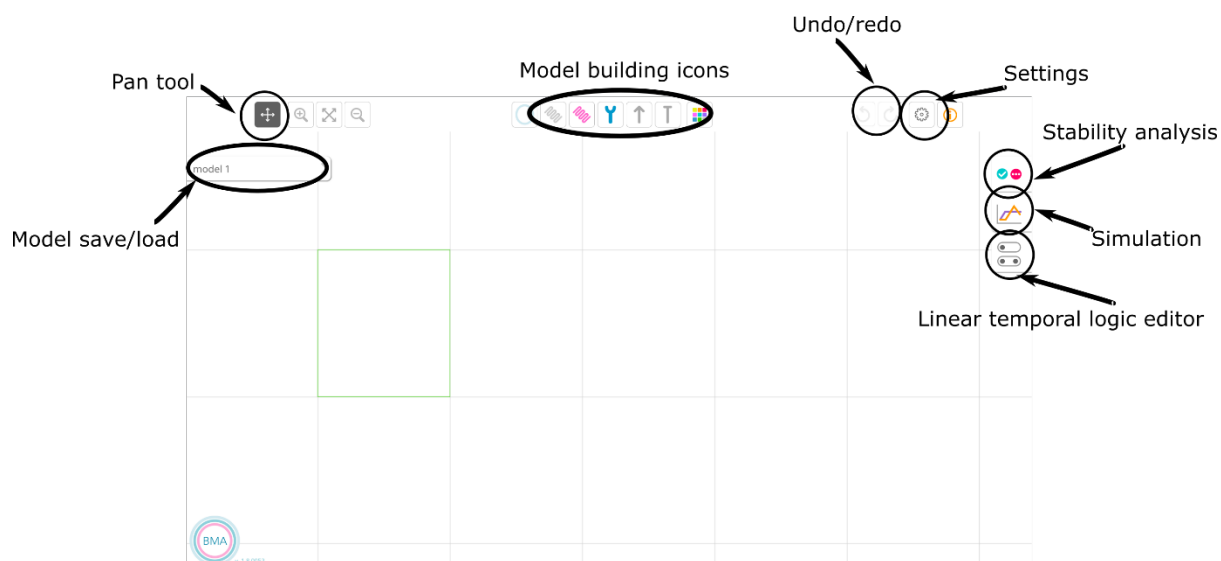
Introduction

This practical deals with understanding biological networks by modelling them as Boolean networks and qualitative networks. In it you will construct different networks and analyse their properties. Here we primarily focus on the use of BioModelAnalyzer (BMA), whilst also introducing GINsim for comparison. BMA and GINsim have similar aims, but different algorithms for modelling and analysing systems, as well as different ways of describing how networks update and change over time. Outside of the differences in the user interface, the key difference between the main tools are that;

- BMA focuses on *synchronous* systems- models where all variables update simultaneously
- Updates in BMA are determined by a single algebraic function on dependent variables

The goal of this practical is for you to familiarise yourself with the modelling techniques and the different strengths and approaches for each tool, whilst introducing different example workflows. In the first and second sections we will study specific biological motifs using different types of model, and both tools. In the third section we will explore more complex, realistic models and analyse them. In the fourth section we will illustrate how to reuse motifs created in the BMA. Questions in each section are intended to reinforce your knowledge from the lectures and your understanding of the practical.

Guide to the BMA user interface



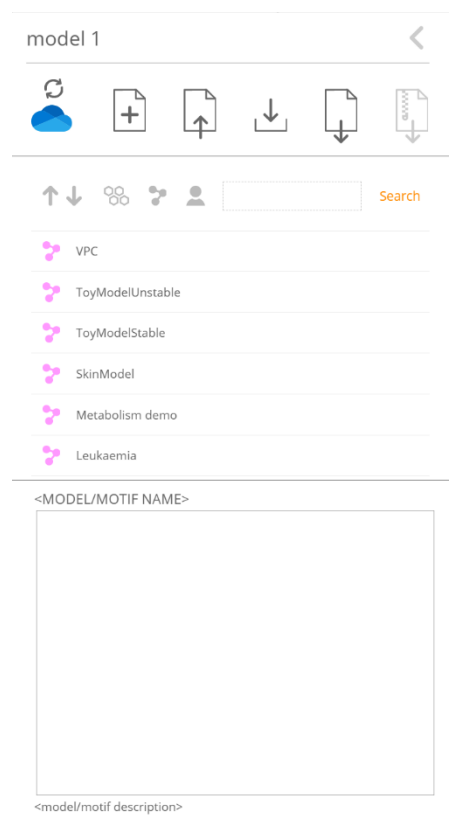
The BMA user interface consists of a blank canvas onto which you can build models and run analyses. In this section we will run through some of the basics of opening and navigating models .

Open a model

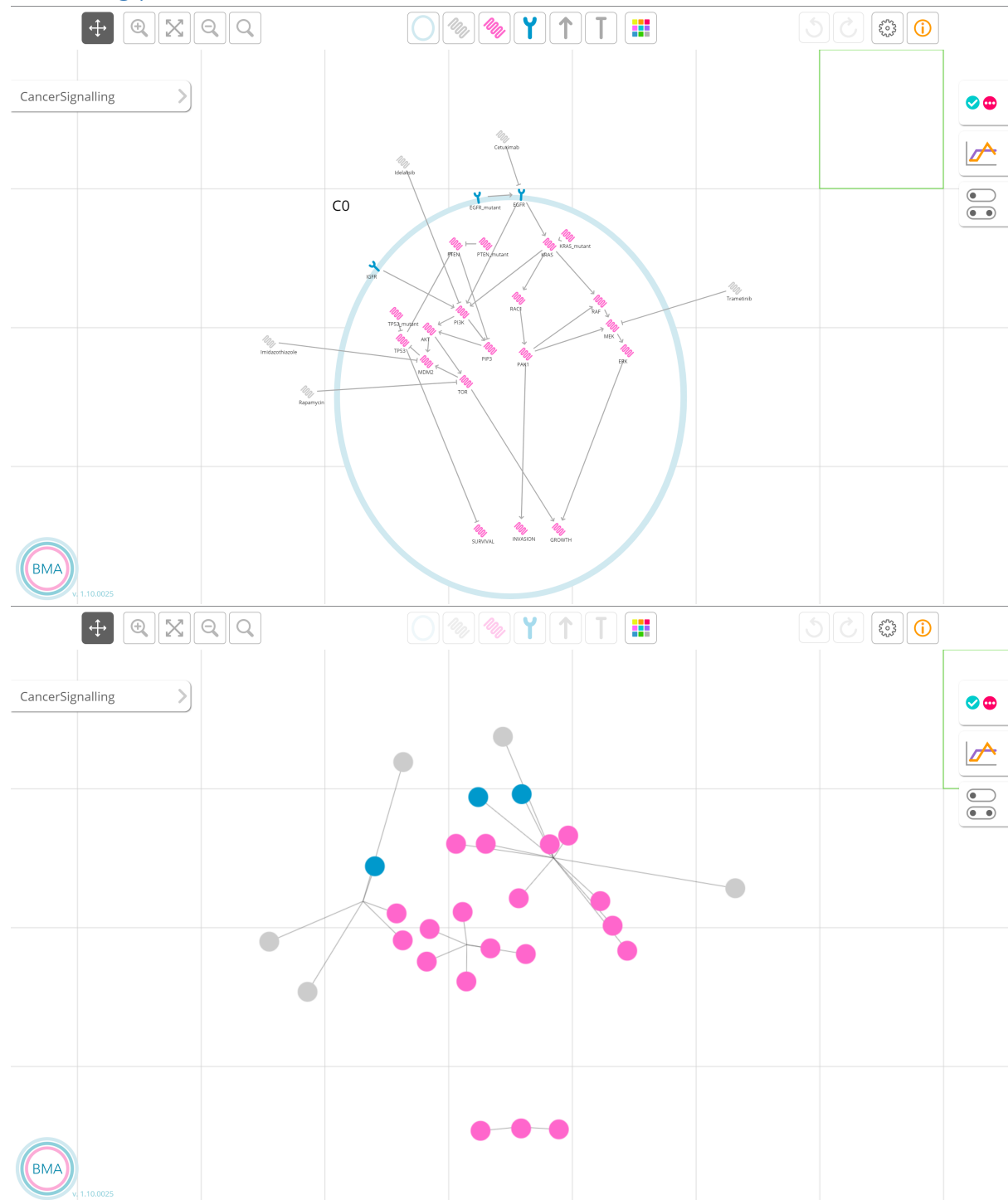
To start learning the tool, we will open a previously built model and run some analysis on it. The BMA tool makes several published models available for you to open and explore, built into the user interface. Here we will explore one demo model that represents a simplified view of signalling in cancer, and analyse the model for stability.

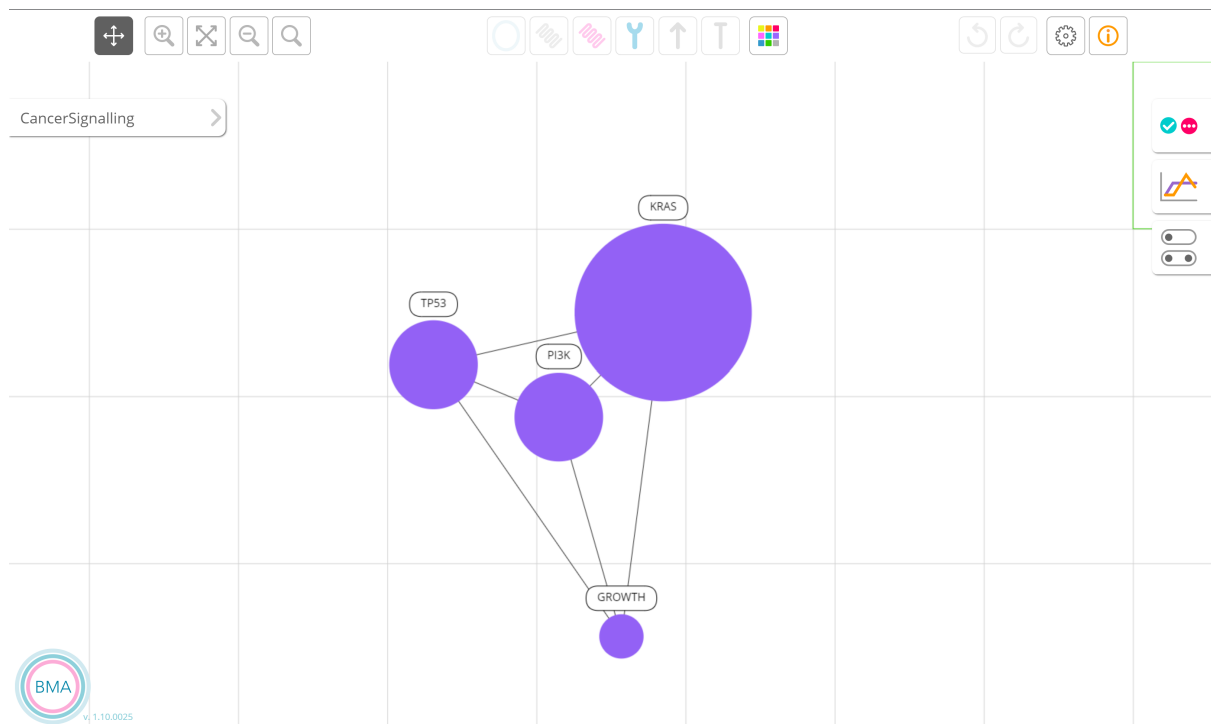
To open models click the arrow on the model save/load icon and a panel will open revealing several controls. All user created models saved in the tool are presented here, alongside built-in models and motifs. User models are distinguished using the user icon, and built-in models can be hidden by clicking the user icon in the repository filtering and search tools. By default, user models are stored in the browser cache (immediately above the model list). Alternatively users can log into a onedrive account and store their models in a folder on the cloud.

To find the cancer signalling model, either scroll down until you see the name “CancerSignalling” or search by typing into the text box. To preview the model, click on the name. This shows a small rendering of the model in the lower panel. To open it double click on the name. Click on the arrow to close the model repository panel.



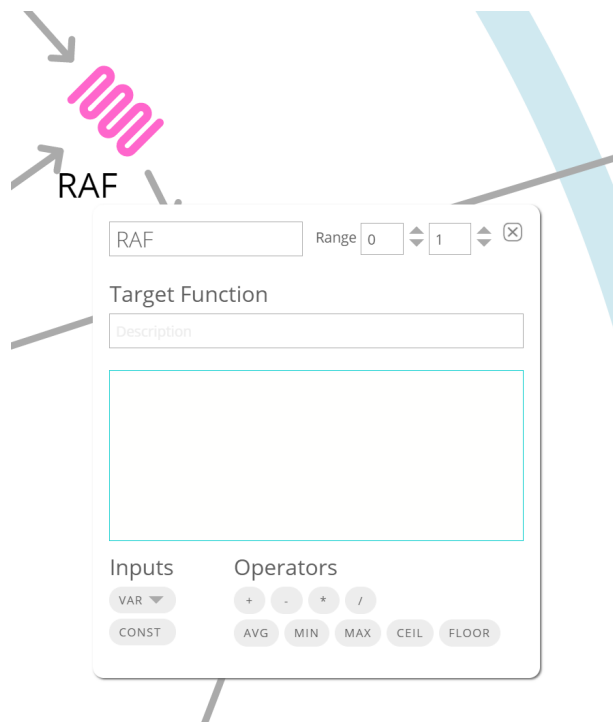
Visualising your model



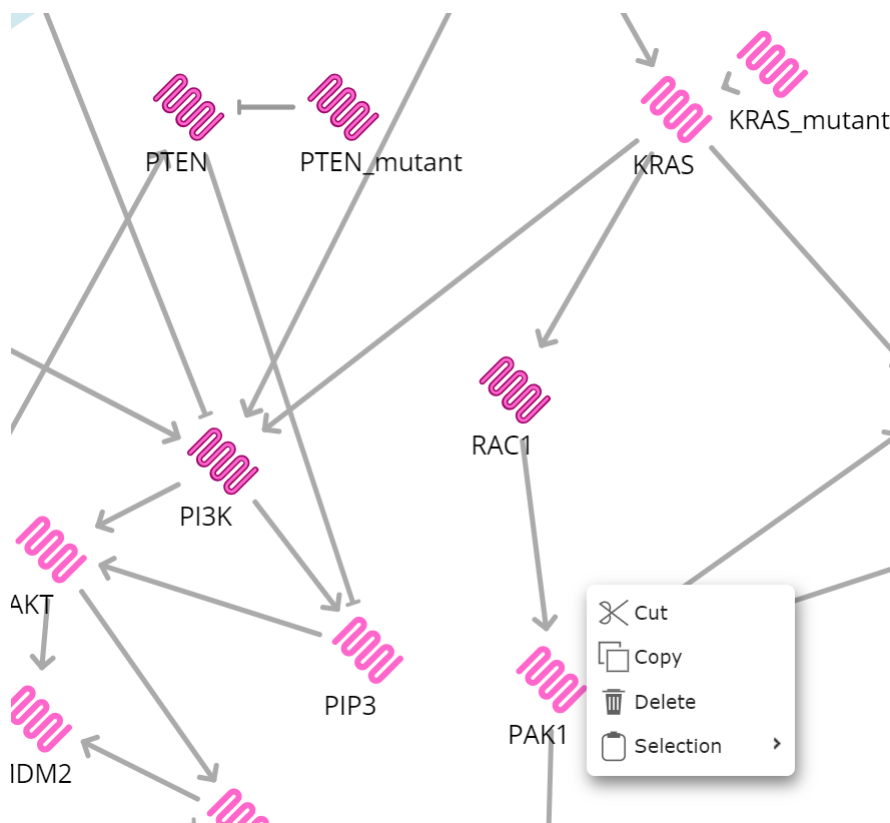


To explore the model, use the controls in the upper left. The zoom buttons or mouse wheel can be used to zoom in and out. When the pan tool is selected, click and drag on the canvas to move around the model. Finally, the model can be fitted to the screen. Large models are by default visualised with an “intelligent zoom”, where variables are replaced by clusters as the user switches to a far zoom. This can be manually disabled in the settings menu. Finally, individual variable names can be searched for using the open magnifying glass button.

Variables are rendered using icons representing membrane bound, intracellular and extracellular proteins. Activations and inhibitions are represented by pointed and flat arrows. Finally cells are drawn as blue circles that cover between one and nine grid spaces.



The variable ranges and update functions can be viewed by right clicking on a variable and selecting edit. Two text boxes allow users to write a description of the function (upper box) and to customise the target function. The default (blank) function is the average of the positive inputs minus the average of the negative inputs. Automatic support (including Intellisense highlighting and autocomplete) support writing functions.



Finally, variables can be selected whilst in navigation-mode (selected using the compass icon, upperleft) by right clicking and dragging (or shift-left clicking and dragging). Selected variables are highlighted. The selection can then be recoloured, using the palette icon, to highlight important variables or pathways in the network. They can also be copied to the clipboard, or used to save the selection as a motif in the model repository through the selection dialogue box. Click on the pan icon to clear the selection.

Running GINsim

GINsim depends on Java, installed on your virtual machine by default. To get the tool, download the jar file from

<http://ginsim.org/downloads>

In this tutorial, version 2.4 is used.

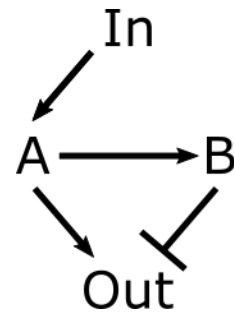
To run it, open a terminal and run java as below

```
java -jar ginsim.jar
```

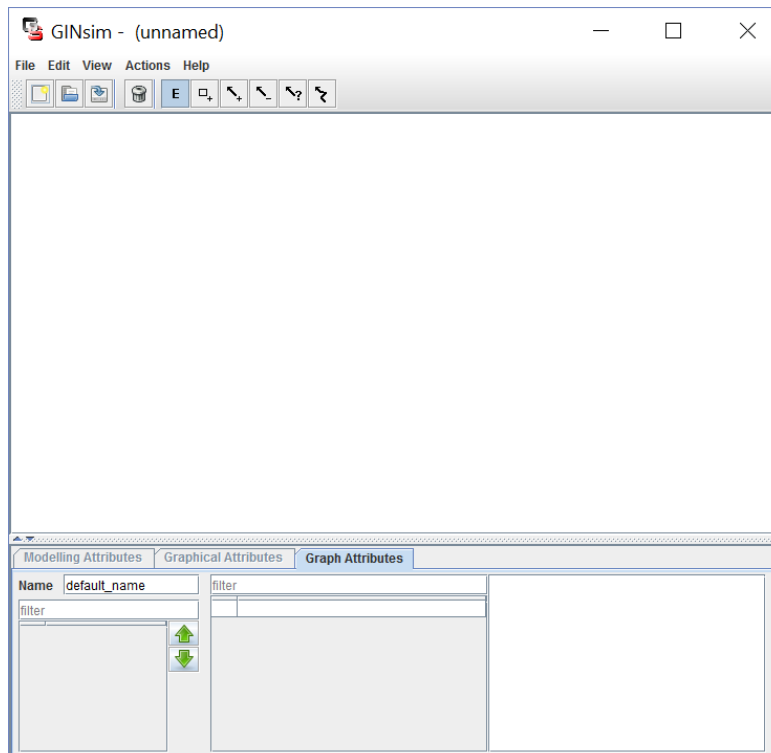
Boolean networks


The first network motif we will analyse is a negative feed forward loop (Figure 1). This allows for perfect adaption to an input, with a transient activation of the output variable (Tyson, Chen & Novak 2003, [http://dx.doi.org/10.1016/S0955-0674\(03\)00017-6](http://dx.doi.org/10.1016/S0955-0674(03)00017-6)). That is to say, if the input is constantly high or low, the output eventually becomes low. However, if the input is low and becomes high, the output rises then falls.



If each variable is Boolean value (i.e. equal to zero or one), how many states are there in the whole system?



We can construct this network in GINsim and BMA to explore how the network responds to different constant inputs. Initially we will use GINsim to construct the model. The window should appear as below.



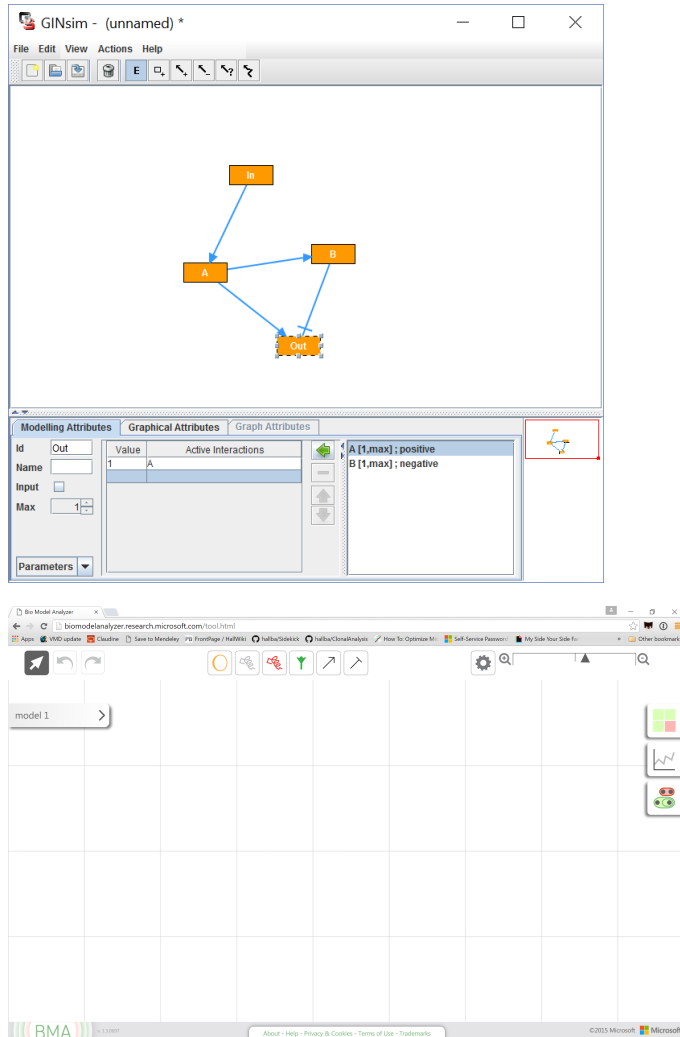
To add nodes (variables such as genes or proteins), click the  button. Similarly to add

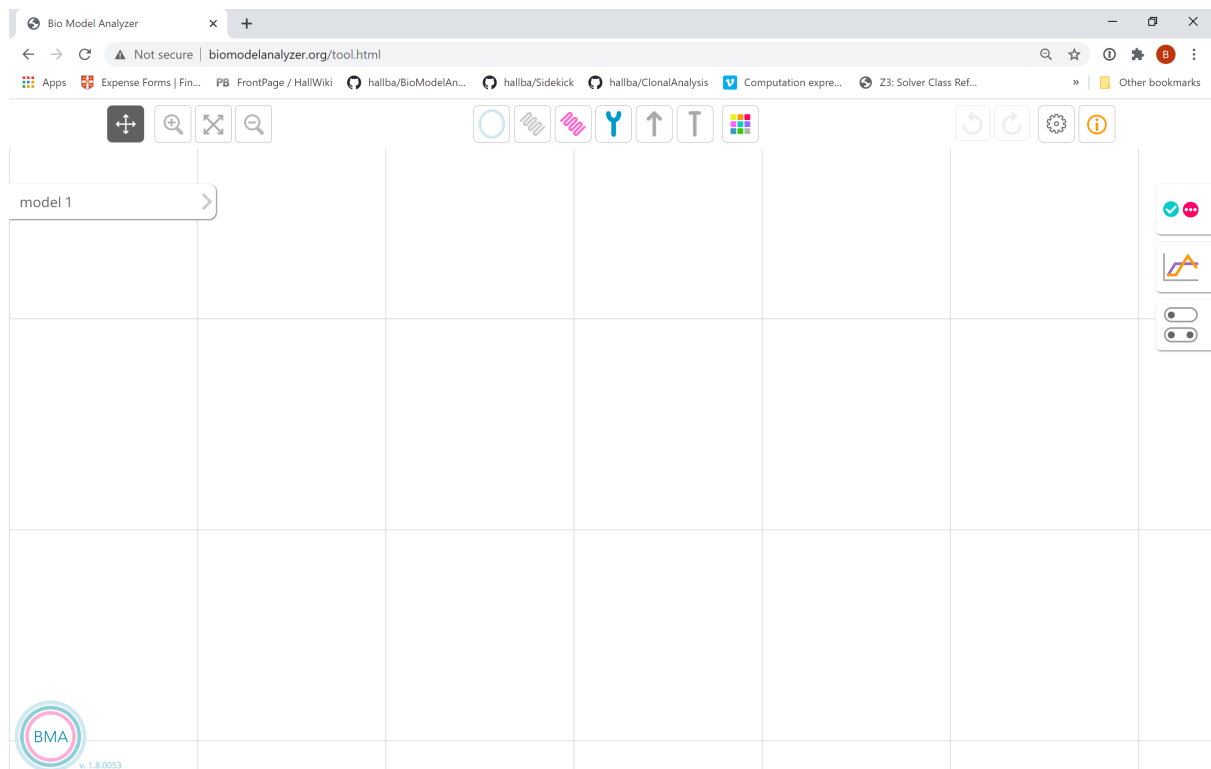
connectivity in the network, click either the activatory  or inhibitory  buttons and draw to connect the nodes. Node names can be changed by clicking on a node and changing the node Id in the "Modelling Attributes" tab (bottom of screen).

To finalise the model, you need to specify the functions on each node; these describe how the value of the node changes depending on its inputs. For nodes with a single incoming connection, this can be specified in "Modelling Attributes". Select the incoming node and click the green arrow, and select 1 as the value that you want the node to become on activation (i.e. if the protein is activated, it turns on). If you don't specify a rule, it will tend to zero.

For Out, you want to specify that if A is active but B isn't $A=1$, but under other circumstances ($A=0$, or $A=1$ and $B=1$) $Out = 0$. As the functions default to zero, you only need to select $A=1$ activation; in more complex cases you may need to pick specific values for each combination, or specify the behaviour with a logical formula.

For In you can choose a basal value by clicking the green arrow without any input selected, allowing you to choose constant high or low inputs. For now, leave the basal value as zero.





We can construct this network in BMA to explore how the network responds to different constant inputs. First add a cell, then add (pink) intracellular variables and connect them using appropriate arrows. Functions describing how variables change in BMA are specified by a target function that can be found by right clicking the variable. By default, this is the difference of the average of positive inputs and the average of the negative inputs; as such these do not need to be edited for this model. The input value can be specified as a constant by writing its value as the target function (i.e. typing 1 or 0 into the target function box). Again, leave this value as zero for now.

Analysing models

The negative feed forward motif is an example of “perfect adaption”. That is to say, for any constant input (zero or one), the system always returns to a single state regardless of the initial state (and not a loop), and in that state “Out” is zero (regardless of whether the input is 1 or 0). A system that is in the inactive state that is exposed to a high input will respond by activating the “Out” variable transiently.

In GINsim we do this by calculating the state transition graph explicitly. To do this, go to actions and click run simulation. In the new dialog, select synchronous under “Construction strategy” and click run. This will return a graph, which you can display as states (specific assignments of each variable) and arrows that show the following state. Clicking on individual states will show in a table at the bottom of the window the specific variable assignments.

How many states have no outgoing transitions? (Hint- they appear as bubbles rather than boxes)

Based on the expected behaviour above, how many states should have no outgoing transitions? Does the model fit the expected behaviour?

What happens when you change the basal activity of In to 1? Does the model still behave as you would expect, and are there other features that you can study to confirm the model behaviour?

The BMA, in contrast to GINSim, does not calculate the state-transition graph explicitly. Instead it can prove that a system is stable- that is, there exists a single state with no outgoing transitions, and no cycles- using a proof that does not require the graph to be explicitly calculated. You can attempt



to prove stability with the button. This will attempt to prove stability. This method works in multiple steps. An initial, quick step is capable of proving stability in isolation for many systems. If it does not you can do a slower check (“further testing”) that will either prove stability or show you why the model is unstable (i.e. either multiple end points or cycles). This test is performed in the backend using Z3.

Does the model behave as you expect in BMA?

To test the transient behaviour in the BMA, there are two options. The first is to perform a



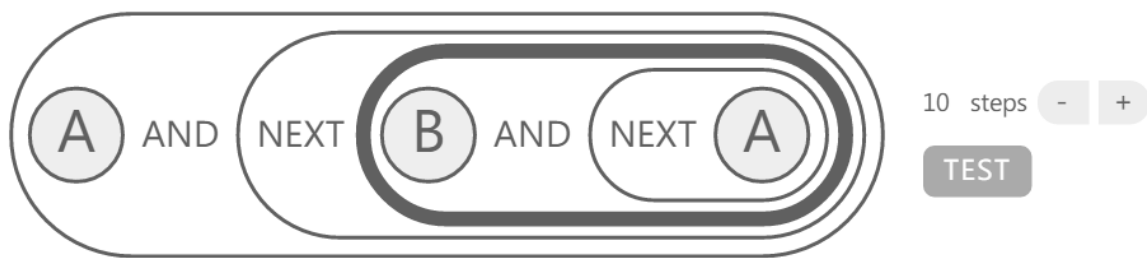
simulation from a specific state and observe how the model changes over time. This button runs simulations. This works well if you know the initial state, or only wish to search a few states.

To test the transient activation, what would be a sensible starting state? Can you show that the model reproduces the behaviour in BMA using the simulation tool alone?

An alternative when you do not know the exact state is to describe the simulation you want to see as a formula, and search the model for simulations that match it. This can be done in the BMA



though a graphical LTL formula builder accessible with the button . To search the model you can define “states”- sets of variable assignments or restrictions- and “queries”- the formula that describes the order and relationship of the states with one another. *States are named alphabetically on generation, and should not be confused with variables of the same name!* When the query is tested, it returns whether some, all, or none of the simulations satisfy the query, and the user can view examples of simulations. For this transient behaviour, create two states with “Out” = 0 and 1 (hereafter referred to as A and B respectively). Then construct a query via drag and drop of A and next (B and next A) to find a simulation where “Out”=0, and in the next step “Out”=1, and in the step following that “Out”=0.

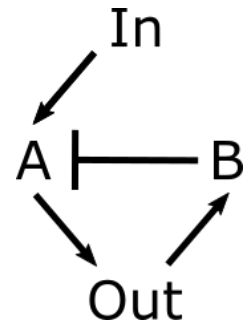


What does the query return if the target function of ln is 1?

What does it return when the target function of ln is 0? Why do you get these results and do they make sense?

Qualitative networks

Here we will construct an alternative motif, a negative feedback loop. In contrast to the negative feedforward loop this topology can have different properties depending on the node properties (either homeostatic or oscillatory, see Tyson 2003). Initially we will construct this motif naively and see what the properties of the network are. We will model the network though as a qualitative network, or multi-state Boolean network. This allows for more detailed models to be created where for instance instead of each variable being on/off each variable could be low/medium/high. Note that the state of each variable can only change by 1 in each step of the simulation, and as such the “function” describes the direction of change (i.e. either the same, up, or down). So if a variable’s value is 0, and its function evaluates to 2, it increases to 1 in the next step. In this part of the exercise, use both GINsim and BMA in parallel to answer the questions.



To create multistate variables in GINsim, construct the network and increase the maximum value of each variable to 2 in modelling attributes (i.e. each variable can be 0, 1, or 2). In order for variables to “see” multiple states from incoming variables, you also need to click on the connecting arrows and add the extra state (in modelling attributes). The functions for each variable except A should equal to the positive incoming value, whilst the function for A should be equal to the difference of In and B. Ensure that “B” and “Out” are sensitive to all input levels of their upstream variables, and make sure that the target value for “A” is equal to “In” minus “B”; you will need to select multiple input states to get this working (i.e. B=1 and In=2 should lead to the level A=1). To do the same in BMA, increase the maximum variable in the target function editor; the default target function should capture the networks behaviour correctly without further information.

Finding loops in GINsim becomes harder as model complexity increases using the state-transition graph alone. Whilst it is easy to identify fix points, visualisation of oscillations in large graphs is harder as it requires searching by eye. To simplify this, the state transition graph can be analysed to find strongly connected components- that is a set of states that revisit one another. To do this go from the state-transition graph and go to actions and select SCC graph. Then run the analysis and oscillations (as strongly connected components) will be redrawn as fixpoints.

There are two ways to find loops in BMA. In the stability proving tool, if a loop exists it will be found by using the further testing button. However, this will only find the smallest loop- if there is more than one loop and you want to identify all of the loops, this must be done by using the LTL interface. The query “eventually oscillation” will find an example of a loop; to find alternatives, go to the example, right click the last timepoint in the table and select “Create LTL state” (hereafter referred to as “A”). Then to find a different loop, use the query “always not A and eventually oscillation”. This can be repeated for alternative loops (e.g. “always not A or B and eventually oscillation”) until the query returns “True for none”. Note that this will only find loops up to the length of the search, indicated alongside the query.

If the basal level of In is zero, is the model stable?

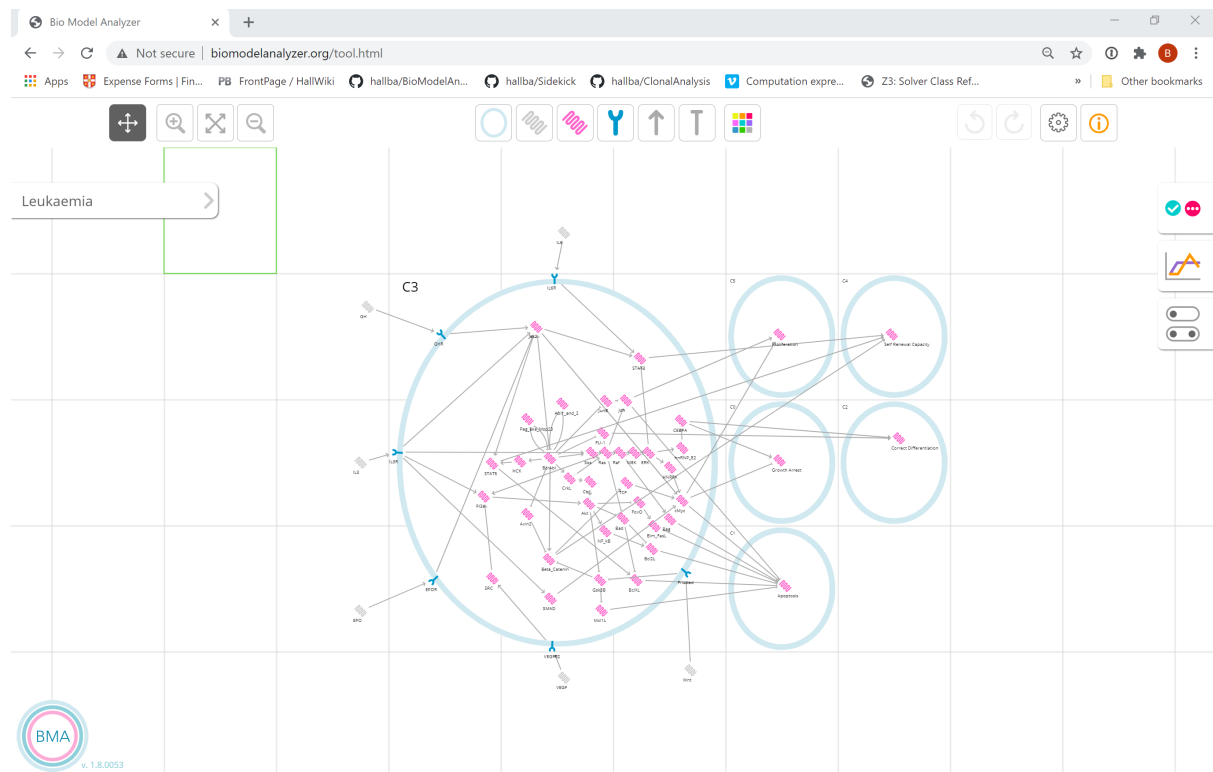
What about if the basal level of In is one?

How do the endpoints change with different basal levels of In?

What constant levels of I_n give a stable network, and which motif do you think this network corresponds to in Tyson's paper?

Complex networks

Here on we will use the BMA to analyse a large network and explore some of its properties. We are studying the properties of a model of chronic myelogenous leukaemia (CML) from “Drug Target Optimization in Chronic Myeloid Leukemia Using Innovative Computational Platform” by Chuang et al (Scientific Reports, 5:8190, Nature Publishing Group, February 2015). This model has 54 nodes, each with a granularity of 3 (i.e. there are 3^{54} or $\sim 10^{25}$ states). The size of state space illustrates the strength of the BMA algorithms here; despite the large number of states both stability proving and LTL analysis is possible. This model is available in the user interface through the built-in model library (named “Leukaemia”).



The model represents early stages CML, where the disease is relatively controllable. Progression to more dangerous stages of CML (chronic phase and later blast crisis) is characterised by the creation of the promiscuous and constitutively active kinase BcrAbl by a chromosomal fusion event. This leads to B-catenin activation and blast crisis. To model the early stages we have restricted the range of BcrAbl (in the centre of the system) to 0. This model is available to users in the model library under the name “leukaemia”. To open the model, find the model with that name and double click.

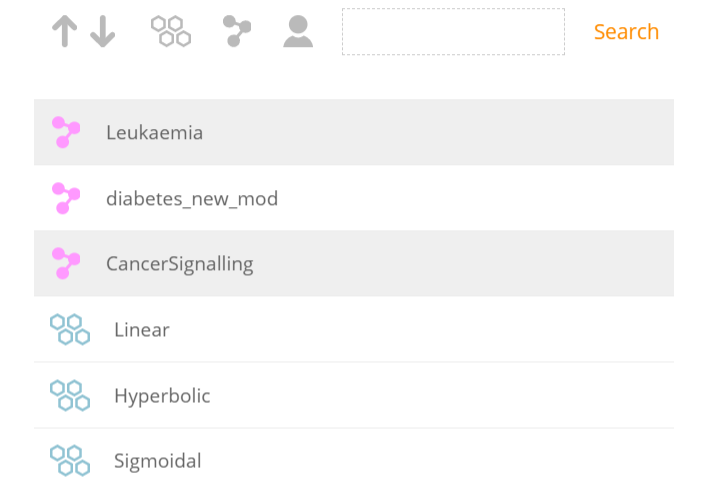
How would you investigate the effects of BcrAbl and B-catenin activation?

A drug (Gleevec) inhibits BcrAbl, and can be modelled by subtracting 1 from the target function of BcrAbl. This introduces an instability into the model- describe this instability. What would this instability do to the cell behaviour on drugging?

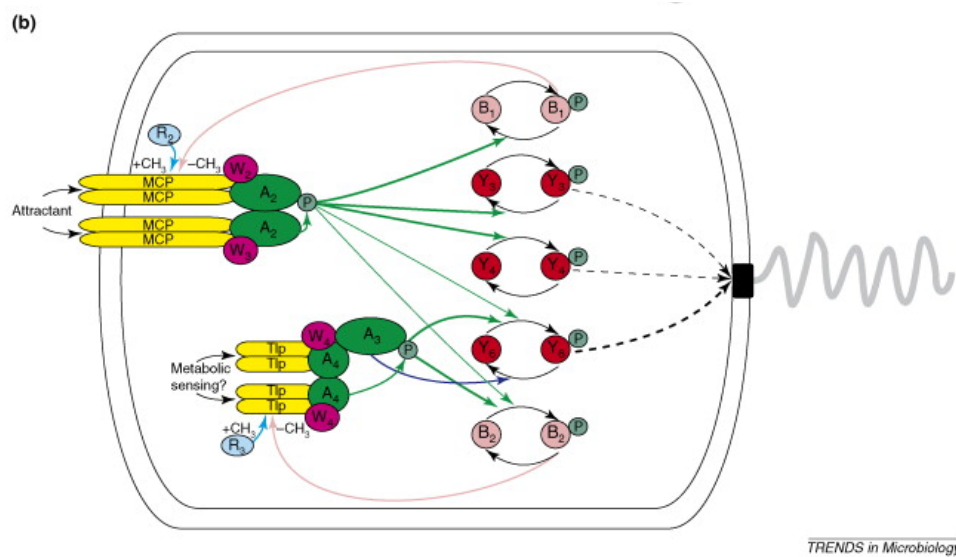
An effective drug strategy may remove this instability to encourage different types of behaviour. Use the LTL interface to study how a drugged cell ends in different fix point or cycle states, and suggest how this might be changed.

Model construction from component motifs

BMA has been designed to enable model reuse and combination. In addition to opening a model in the canvas, models can be dragged from the preview window and added onto the working model in the canvas. This allows common network components to be reused as motifs in a model.



Users can create new motifs by creating a selection, right clicking and picking “create motif” from the selection menu. Motifs are tagged in the model repository using a blue “honeycomb” icon, and both models and motifs can be filtered using the organising and search tools at the top of the repository.



constant. For any constant value of “I”, the model is stable (and can be proved so using the stability checker, sometimes with further testing).

Rename the variables in this motif to match the names of the MCP driven cluster;

I → MCP

A → A2

B → B1

O → Motor

Target functions in related variables should change their names automatically.

Drag a second homeostasis motif to the screen, and rename the variables to match the internal state cluster;

I → Metabolism

A → A4

B → B2

Finally, remove the second O and connect A4 to the motor variable. Change the target function to represent an average of both CheA molecules;

$\text{avg}(\text{ceil}(\text{var}(\text{A2})/3), \text{ceil}(\text{var}(\text{A4})/3))$

Test this model with different constants on MCP and Metabolism.

Further reading and resources

Tutorials and examples

Online BMA tutorial

<http://biomodelanalyzer.org/Tutorial.pdf>

Constructing and Analyzing Computational Models of Cell Signaling with BioModelAnalyzer.

Benjamin A. Hall, Jasmin Fisher **Current Protocols in Bioinformatics** 2020

<https://doi.org/10.1002/cpbi.95>

Schools materials (age 16-18)

Working with the UK exam board OCR, we have prepared some materials for A-level student to learn about cancer biology using the BMA.

Introductory blog:

<https://www.ocr.org.uk/blog/getting-started-with-computational-biology-in-cancer-research/>

Worksheets:

<https://biomodelanalyzer.org/Teachers%20sheet%20v3.docx>

<https://biomodelanalyzer.org/Student%20sheet%20v3.2.docx>

Introduction to Z3

Introduction

This is intended as a basic introduction to using Z3 to solve compute answers to questions. Z3 is a *SMT solver*- one of a class of tools that allows you to identify find solutions to problems from their definition. In contrast to typical programming problems, where you specify an algorithm, in Z3 and other SMT solvers you define variables (integers, Booleans, or others) and the relationships between them. This is used as input to Z3, which finds a solution in terms of a specific set of variable assignments if one is available.

In this practical we work through three examples of Z3. The first two focus on illustrating how you can express problems in logic. The third gives a specific example of how to prove stability using Z3 alone, similar to how further testing works in the BMA. Brief descriptions of the notebooks are given below, but the notebooks themselves contain more details.

Technical notes

Each example has a dedicated jupyter notebook with detailed notes, all available at <https://github.com/hallba/WritingSimulators>. Clone these files from github using the following command;

```
git clone https://github.com/hallba/WritingSimulators.git
```

Notebooks are written in F# and download Z3 from github as part of the script. In this practical we run the F# code through docker in two steps. First, enable the user to run docker with the following command;

```
sudo usermod -a -G docker ${USER}
```

and log out and back in again. Secondly, run jupyter using the F# kernel using;

```
docker run -v $PWD:/notebooks -p 8888:8888 fsprojects/ifsharp
```

Sometimes the z3 download process may fail due to local install issues. The download process occurs in the first cell of the notebook. If this process fails, download z3 manually using the links in this cell, unpack the compressed file, and move the folder (named “z3-version-architecture-operating_system.tar.gz”) to the notebook folder. Finally, rename it “z3”. If this is done correctly the first and second cells should run without issue.

Simple example

This example introduces variable and constraint creation with trivial examples to illustrate the results that can be generated using Z3. A and B are integers and are either equal to 1 or 2. In the example we add simple constraints that both give solutions ($A > B$) and cannot be solved ($A = 1$ and $A = 2$).

Timing example

This example builds on the previous one by showing how problems that involve “real world” variables that change over time can be modelled. Using a well-known riddle, popularised in the film “Die Hard with a Vengeance”, you need to work out how to get 4 litres of water using only a 3- and a 5- litre jug, taking steps filling and emptying jugs, and pouring one into the other. In contrast with

the “simple example”, we need to define variables which represent the changing states of entities over time. The solution may not be immediately obvious, but it can be achieved by creating distinct variables representing the amount of water in each jug at each step. For example, the amount of water in each jug initially can be specified by variables named *ThreeJug-0* and *FiveJug-0*. The amount of fluid in each jug in the next step is specified by variables named *ThreeJug-1* and *FiveJug-1*, and constraints determine how the amount of fluid changes between the steps. As we don’t know how many steps it will take to get 4 litres, we programmatically check for solutions, and add new variables and constraints if no solution is found.

Negative feedback Loops

The final example uses Z3 to prove whether a negative feedback loop, implemented as a Boolean network, is stable. In contrast to the BMA, the updates are hard coded in the Z3 constraints for just this network and no fast check is performed. However, the method here of first searching for bifurcations (more than one fix point) and then searching for cycles follows the same approach as used in “further testing” in BMA stability proof.

Further reading and resources

A larger set of tutorials

<https://github.com/hallba/Z3Tutorials>

SAT/SMT solving by example

https://sat-smt.codes/SAT_SMT_by_example.pdf

Programming Z3

<https://theory.stanford.edu/~nikolaj/programmingz3.html>